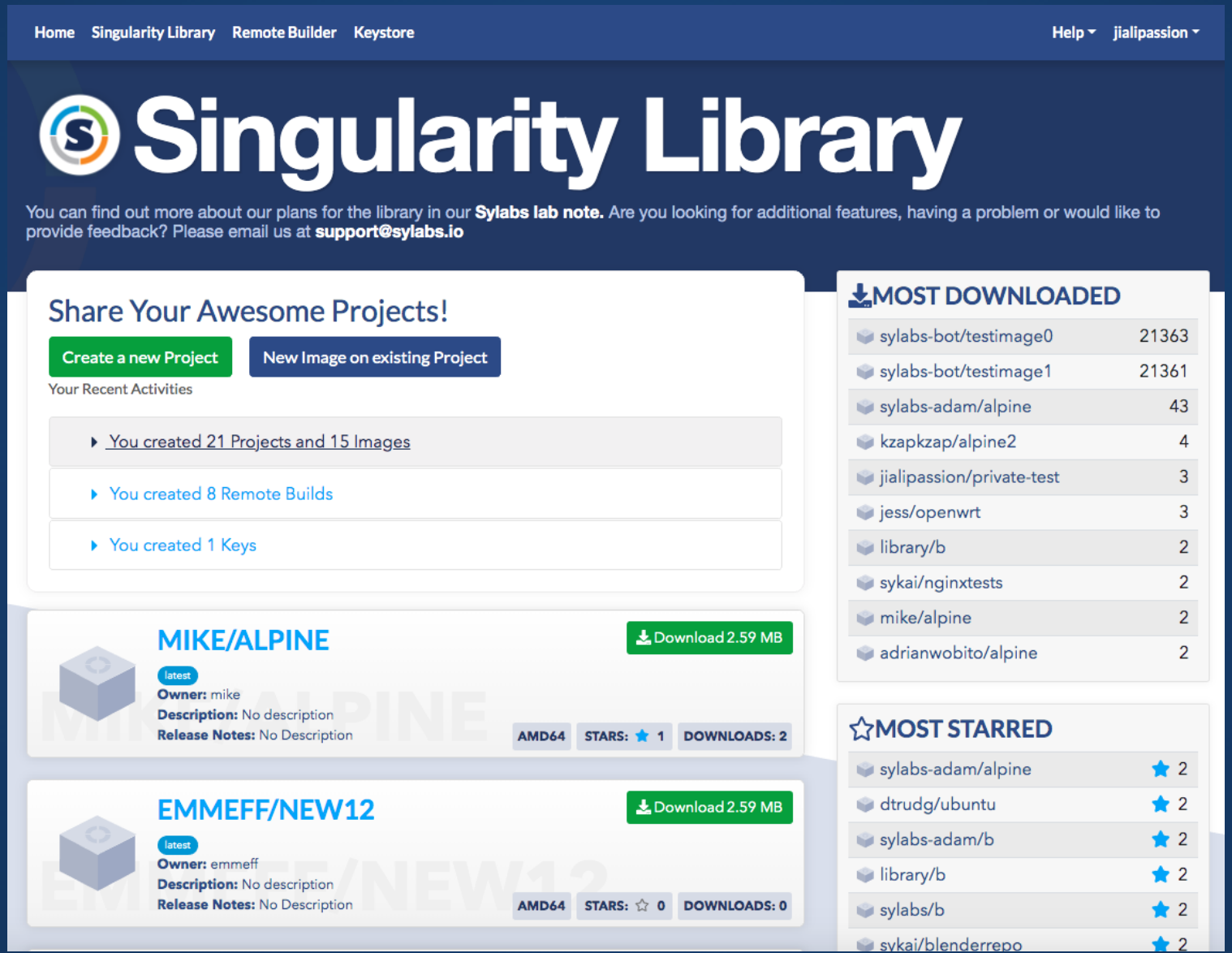
@ngrx/data 进阶之路

JiaLiPassion @ngChina2019

自我介绍

- 李嘉
- 公司: ThisDot
- Zone.js: Code Owner
- Angular: Collaborator
- @JiaLiPassion

我经历的一个真实的项目



Step1：传统的Angular

https://ngrx-data.firebaseapp.com/index.html?auto=false&program=traditional

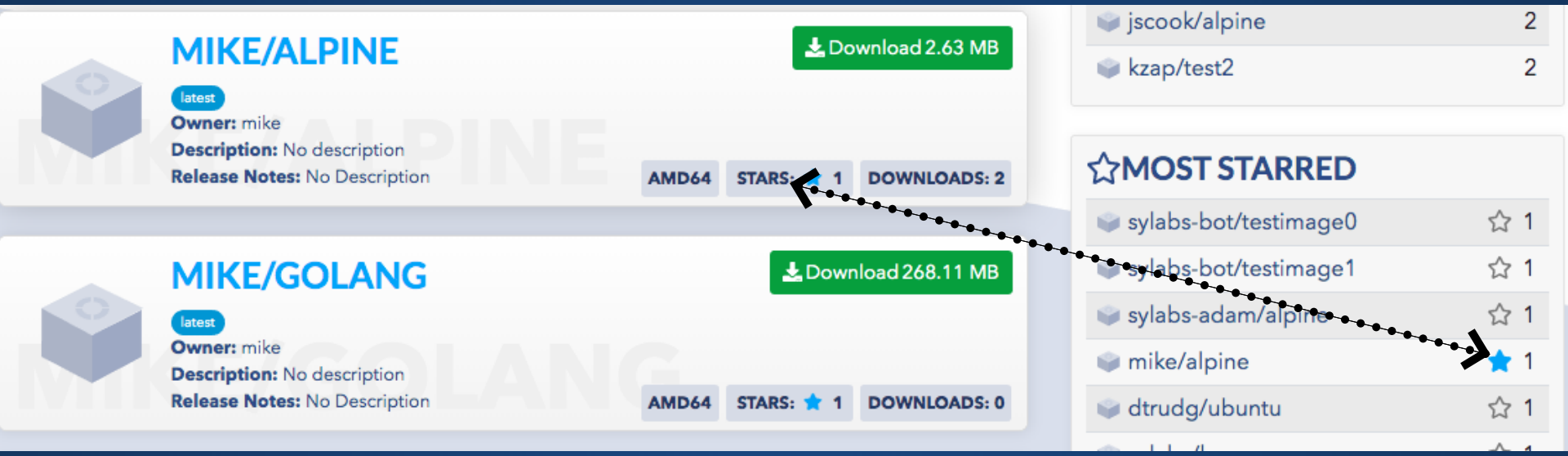
代码示例

```
1  // Container Service
2  @Injectable()
3  export class ContainerService {
4    constructor(private http: HttpClient) {}
5
6    getAllContainers() {
7      return this.http.get('http://service/containers');
8    }
9  }
10
11 // Container Component
12 export class ContainerComponent implements OnInit {
13   containers$: Observable<Container[]>;
14
15   constructor(private containerService: ContainerService)
16
17   ngOnInit() {
18     this.containers$ = this.containerService.getAll();
19   }
20 }
```

代码示例

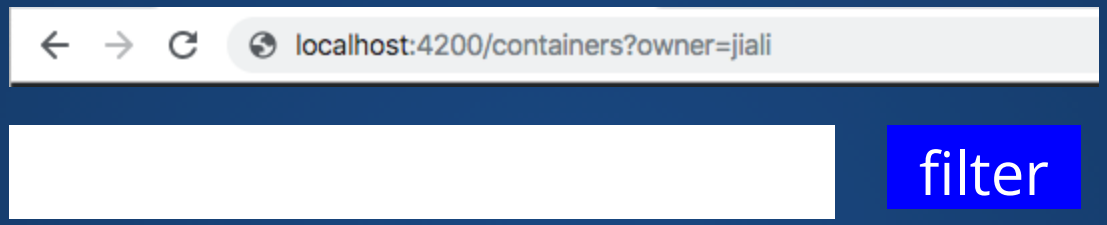
```
1  // Container Service
2  @Injectable()
3  export class ContainerService {
4    constructor(private http: HttpClient) {}
5
6    getAllContainers() {
7      return this.http.get('http://service/containers');
8    }
9  }
10
11 // Container Component
12 export class ContainerComponent implements OnInit {
13   containers$: Observable<Container[]>;
14
15   constructor(private containerService: ContainerService)
16
17   ngOnInit() {
18     this.containers$ = this.containerService.getAll();
19   }
20 }
```

新的需求



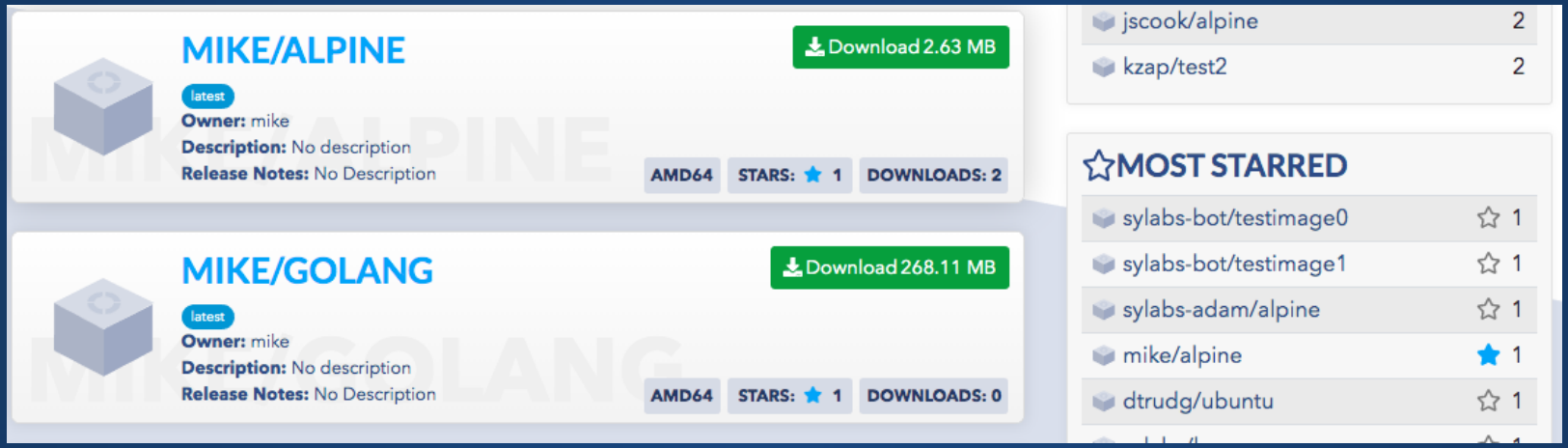
- 可以编辑Image状态

新的需求



- Filter可以联动URL

新的需求


MIKE/ALPINE
Download 2.63 MB
latest
Owner: mike
Description: No description
Release Notes: No Description
AMD64
STARS: 1
DOWNLOADS: 2
MIKE/GOLANG
Download 268.11 MB
latest
Owner: mike
Description: No description
Release Notes: No Description
AMD64
STARS: 1
DOWNLOADS: 0
jscook/alpine 2
kzap/test2 2
MOST STARRED
sylabs-bot/testimage0 1
sylabs-bot/testimage1 1
sylabs-adam/alpine 1
mike/alpine 1
dtrudg/ubuntu 1

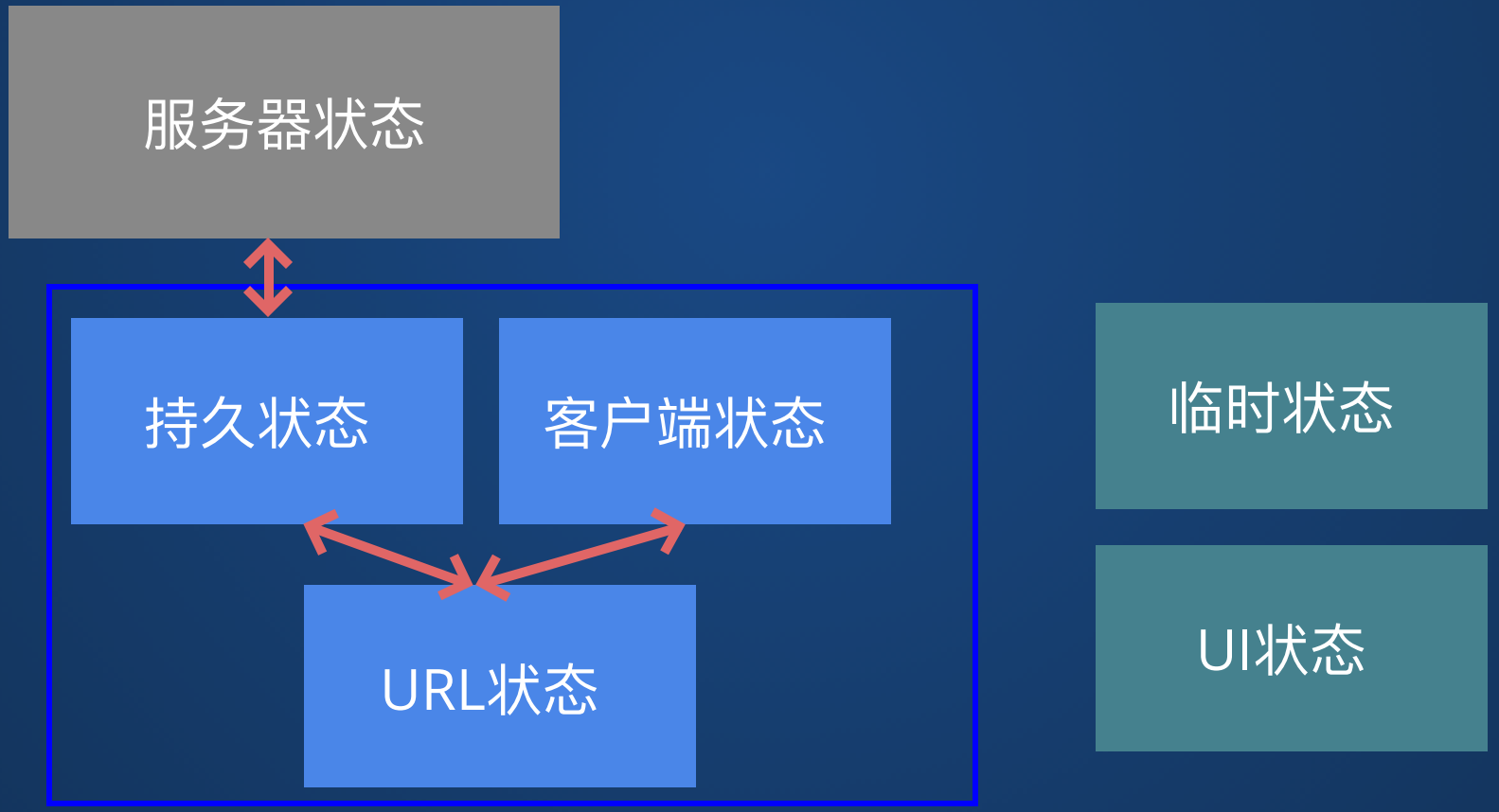
- 支持乐观更新

状态管理

状态

- 服务器端状态
- 客户端状态
  - 持久状态
  - URL/Routing 状态
  - 客户端状态
  - 临时状态
  - UI状态

状态之间关系



持久状态同步

```
lass DashboardComponent {
starChangedInRecentUpdatedComp(starredContainer) {
  updateStarInMostStarredComp(starredContainer);
}

starChangedInMostStarredComp(starredContainer) {
  updateStarInRecentUpdatedComp(starredContainer);
}


lass MostStarredComponent {
@Output() starredComponentUpdated = new EventEmitter<Container>();

starChanged(container) { this.starredComponentUpdated.emit(container)

updateStarInMostStarredComp(starredContainer) { // update local data


lass RecentUpdatedComponent {
...
```

持久状态同步+

```
1  // use a starred service as a bridge
2  export class StarredService {
3    starredContainer$ = new BehaviorSubject(null);
4    starContainer(container: Container) {
5      this.starredContainer$.next(container);
6    }
7  }
8
9  export class RecentUpdatedComponent {
10   ngOnInit() {
11     this.containers$ = combineLatest(
12       this.starredService.starredContainer$,
13       this.containerService.getLatestContainers()).
14         pipe([starredContainer, latestContainers] => {...})
15   }
16 }
17
18 // MostStarredComponent
19 export class ContainerComponent {
20   ngOnInit() { this.containers$ = combineLatest(...)}
```

越来越多的同步需求

Starred

Download Count

Description

Version/Tag Children

Service成了瓶颈

Service成了瓶颈

Starred Service

DownloadCountService

Description Service

Version/Tag ChildrenService

Service成了瓶颈

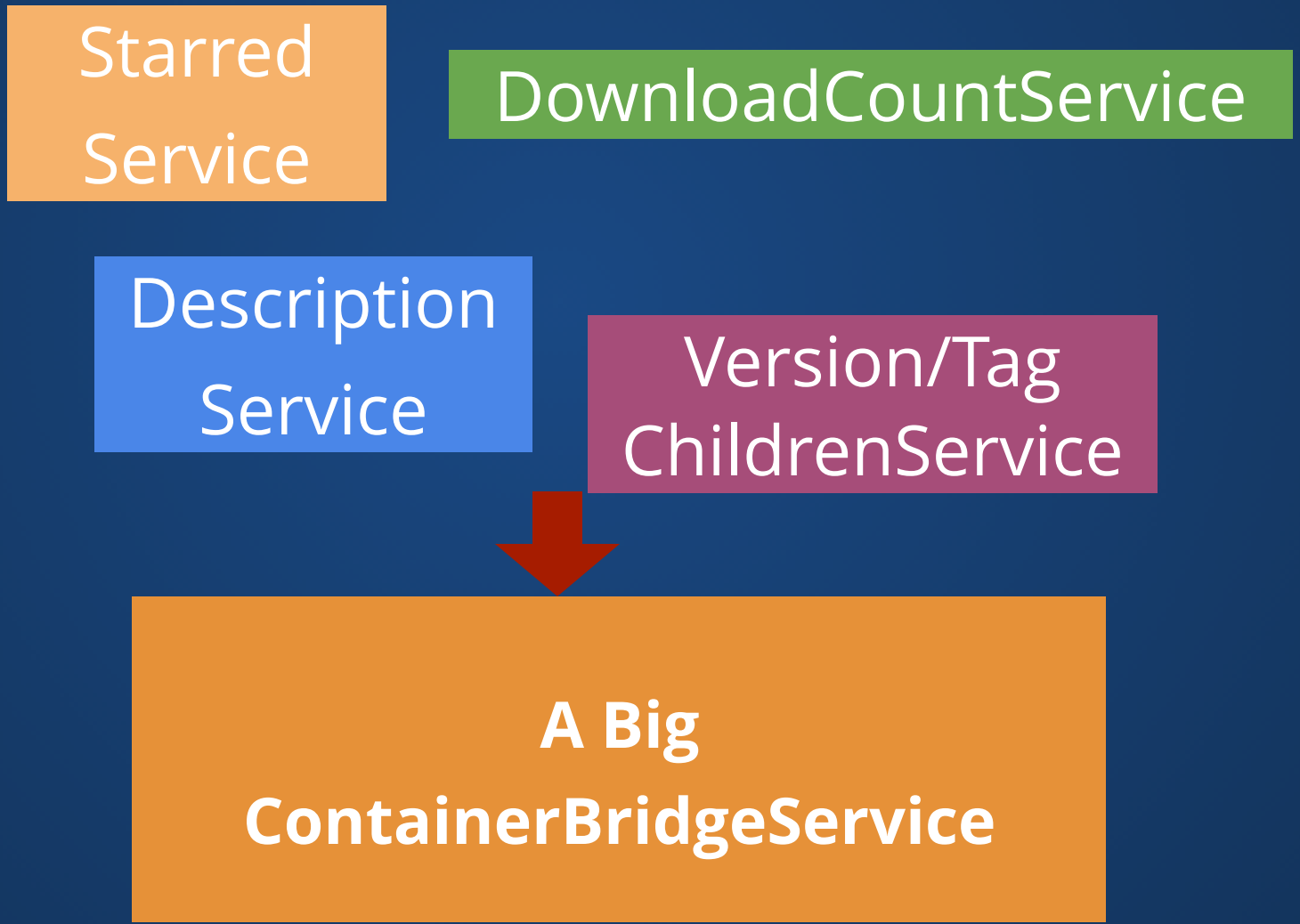
Starred Service

DownloadCountService

Description Service

Version/Tag ChildrenService

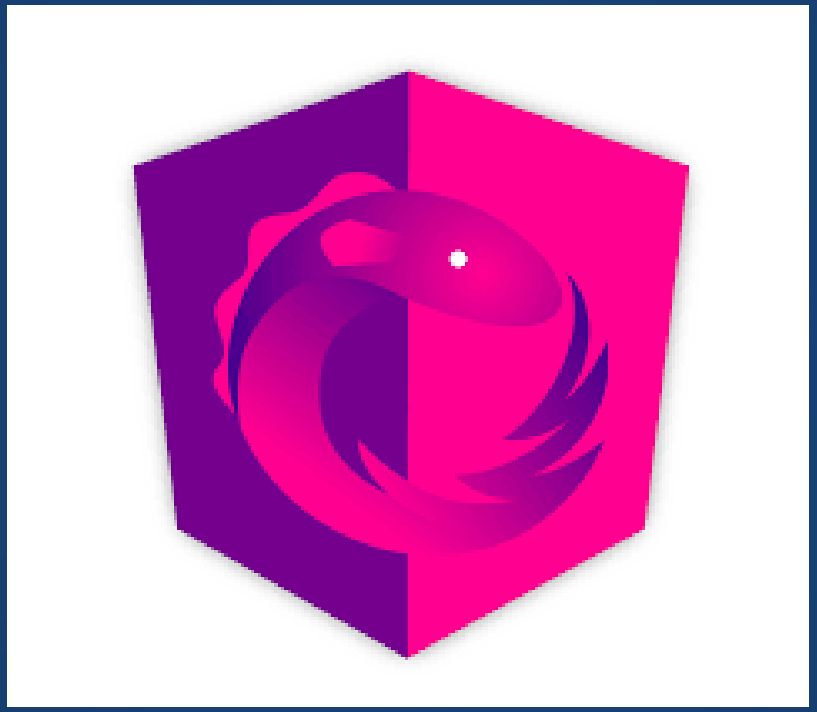
Service成了瓶颈



问题

- 业务逻辑和状态管理混在一起
- Command和Query混在一起
- 各个状态之间的同步没有很好地机制,解决方案治标不治本
  - 持久状态和服务器的状态
  - 持久状态之间
  - 客户端状态和服务器状态和URL状态
  - ...
- 数据是Mutable 可变的
- 各个Component同时更新的处理变得非常复杂

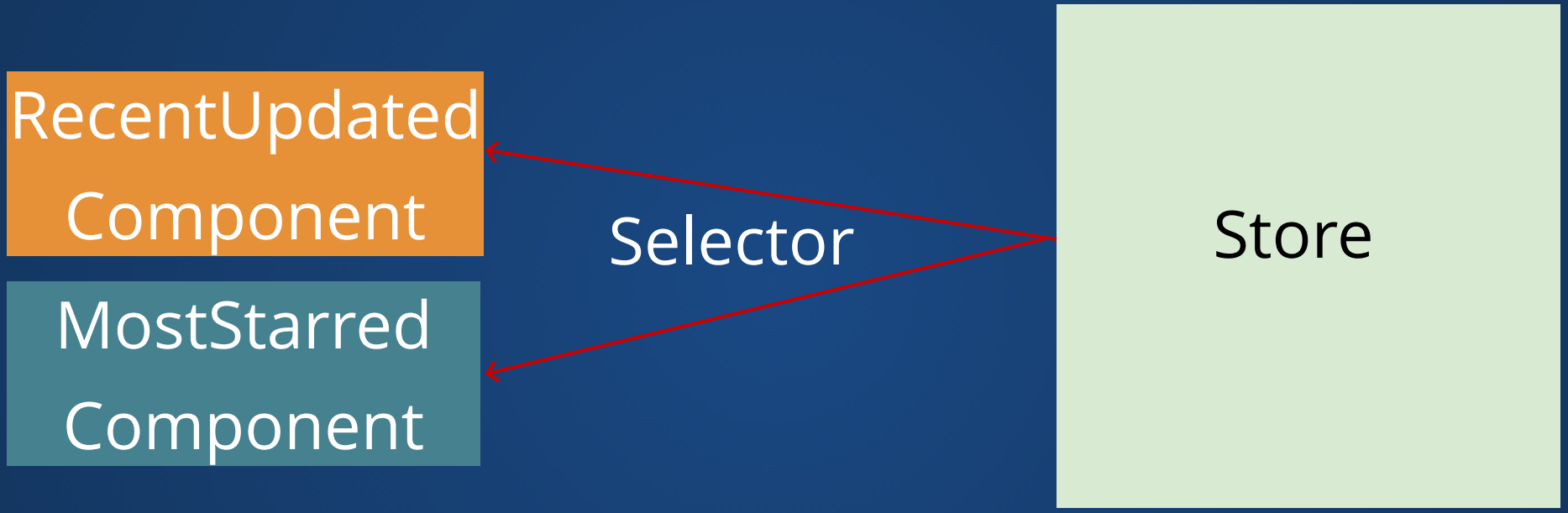
状态管理库ngrx?



NgRx

- 唯一数据来源Store
- 数据只读更好的性能
- Pure Function便于测试
- Command和Query分开,程序便于维护

持久状态同步


RecentUpdated Component
MostStarred Component
Selector
Store

好多层....

https://ngrx-data.firebaseapp.com/index.html?auto=false&program=ngrx

试着实现一个getAll

Actions

```
1  export const GET_ALL = '[Container] query-all';
2  export const GET_ALL_SUCCESS = '[Container] query-all/success';
3  export const GET_ALL_FAILED = '[Container] query-all/failed';
4
5  export class GetAll implements Action {
6    readonly type = GET_ALL;
7  }
8
9  export class GetAllSuccess implements Action {
0    readonly type = GET_ALL_SUCCESS;
1    constructor(public payload: string[]) {}
2  }
3
4  export class GetAllFailed implements Action {
5    readonly type = GET_ALL_FAILED;
6    constructor(public payload: string[]) {}
7  }
```

Reducer

```
1  export interface ContainerState {
2    containers: Container[];
3    filter: string;
4  }
5
6  export const initialState = {
7    containers: [],
8    filter: null
9  }
10
11 export function containerReducer(state: ContainerState = initialState,
12                                    action: ContainerActionTypes) {
13   switch(action.type) {
14     case GET_ALL_SUCCESS:
15       return { ...state, containers: [...action.payload] };
16     default:
17       return state;
18   }
19 }
```

Effects

```
1  export class ContainerEffects {
2      constructor(
3          private store: Store<any>,
4          private actions$: Actions,
5          private containerDataService: ContainerDataService
6      ) {}
7
8      createEffect(() => this.actions$.pipe(
9        ofType(GET_ALL),
10       mergeMap(() => this.containerDataService.getAll()
11         .pipe(
12           map(containers => {return new GetAllSuccess(containers); }),
13           catchError(() => { return new GetAllFailed(); })
14         ))
15     ));
16   }
```

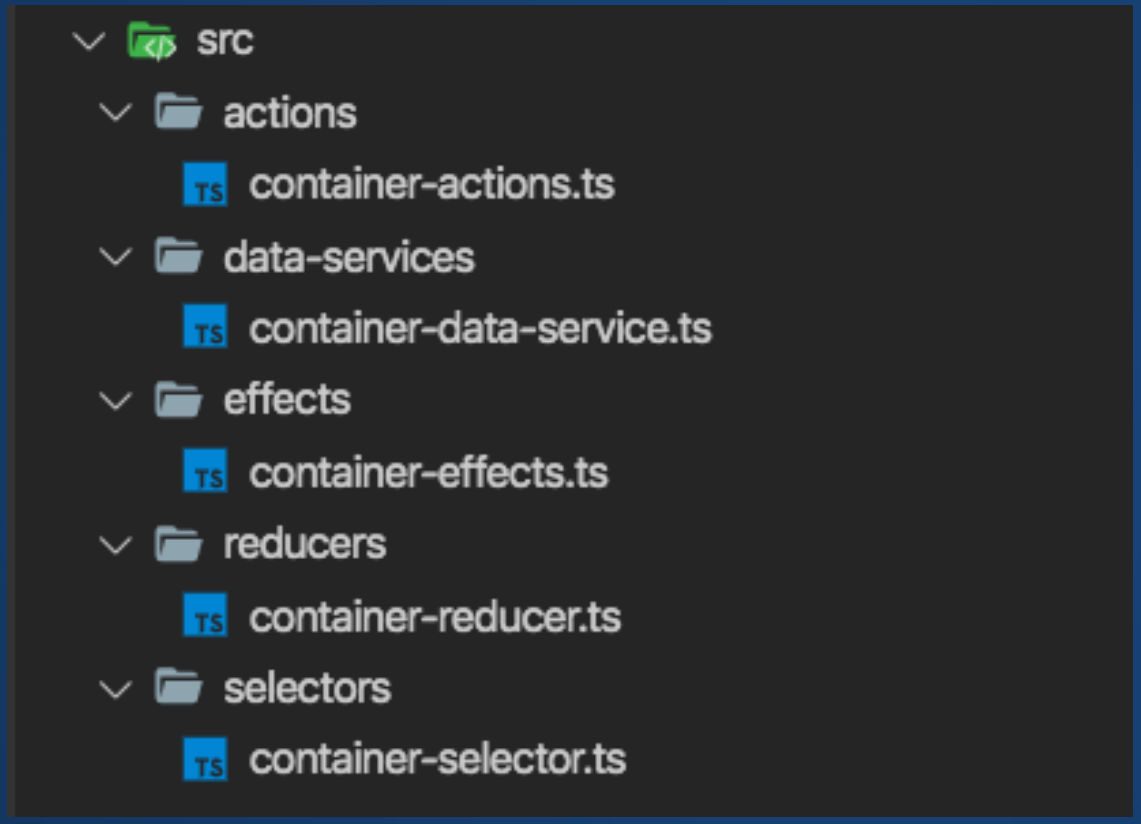
DataService

```
1 class ContainerDataService {
2   getAll() {
3     return this.httpClient.post(...);
4   }
5 }
```

Selector

```
1 export const selectAllContainers =
2     (state: ContainerState) => state.containers;
```

ngrx 构成



10 个业务对
5~10种Actio

好多"Boilerplate"代码

Debug对于初学者不友好

```
1 class ContainerComponent {
2   ngOnInit() {
3     this.store.dispatch(new GetAll());
4     this.containers$ = this.store.pipe(getAllSelector);
5   }
6 }
```

@ngrx/data

Zero Ngrx Boilerplate

你不需要再去实现Actions, Reduers, Effects, DataServices, Selectors.

@ngrx/data

- 实现了对于CRUD 操作
- 支持检索和排序
- 支持加载中和已经加载
- 内置错误处理
- 支持乐观更新(支持回滚/重试)
- 支持所有@ngrx/entity的功能
- 支持所有ngrx的功能

意识不到在使用ngrx

https://ngrx-data.firebaseapp.com/index.html?auto=false&program=ngrxHidden

EntityCollectionService

```
export class ContainerService
       extends EntityCollectionServiceBase {
  getAll();
  add();
  update();
  delete();
  upsert();

  entities$;
  filteredEntities$;
  loading$;
  loaded$;
  ...
}
```

EntityCollectionService

```
export class ContainerService
       extends EntityCollectionServiceBase {
  getAll();
  add();
  update();
  delete();
  upsert();

  entities$;
  filteredEntities$;
  loading$;
  loaded$;
  ...
}
```

entity metadata

```
// define metadata
export const entityMetadata: EntityMetadataMap = {
  Container: {}
};

// provide a facade service
export class ContainerService extends EntityCollectionServiceBase<Containe
  constructor(serviceElementsFactory: EntityCollectionServiceElementsFacto
    super('Container', serviceElementsFactory);
  }
}
```

entity metadata

```
// define metadata
export const entityMetadata: EntityMetadataMap = {
  Container: {}
};

// provide a facade service
export class ContainerService extends EntityCollectionServiceBase<Containe
  constructor(serviceElementsFactory: EntityCollectionServiceElementsFacto
    super('Container', serviceElementsFactory);
  }
}
```

getAll

```
/ call facade from component
export class ContainerComponent {
  ngOnInit() {
    // dispatch action
    this.containerService.getAll();
    // selector to get all entities
    this.containers$ = this.containerService.entities$;
  }
```

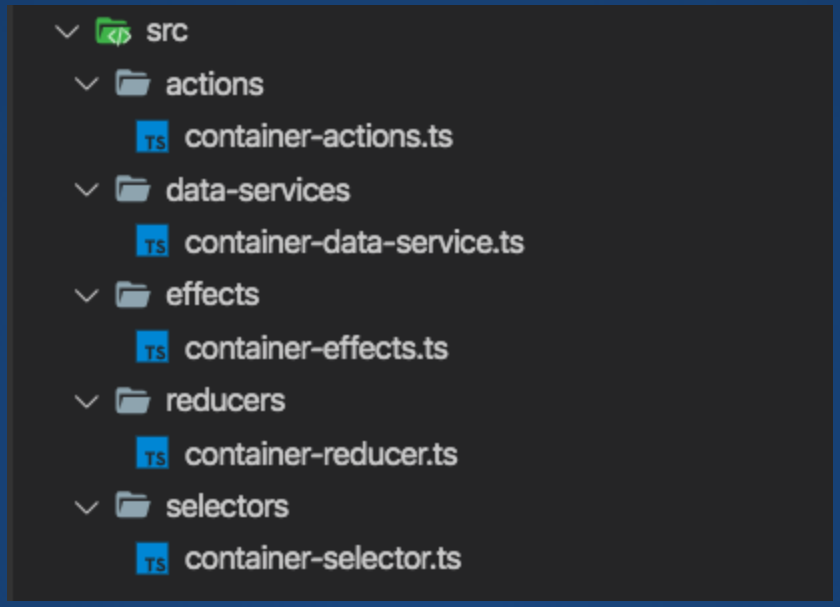
getAll

```
/ call facade from component
export class ContainerComponent {
  ngOnInit() {
    // dispatch action
    this.containerService.getAll();
    // selector to get all entities
    this.containers$ = this.containerService.entities$;
  }
```

getAll

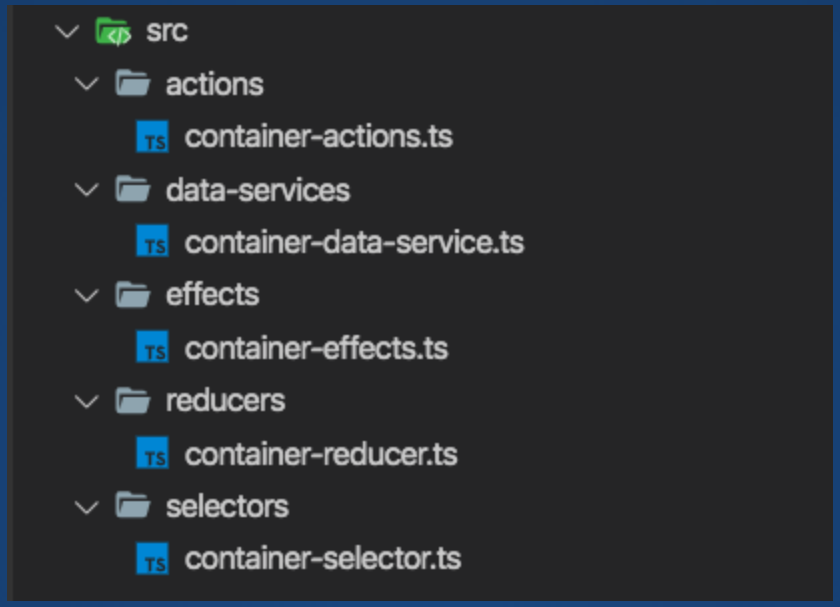
```
/ call facade from component
export class ContainerComponent {
  ngOnInit() {
    // dispatch action
    this.containerService.getAll();
    // selector to get all entities
    this.containers$ = this.containerService.entities$;
  }
```

getAll




```
1 // dispatch action
2 this.containerService.getAll();
3 // selector to get all entities
4 this.containers$ = this.containerService.entities$;
```

getAll




```
1 // dispatch action
2 this.containerService.getAll();
3 // selector to get all entities
4 this.containers$ = this.containerService.entities$;
```

ngrx/data architecture

https://ngrx-data.firebaseapp.com/index.html?auto=false&program=ngrxData

CRUD

```
1  export class ContainersComponent implements OnInit {
2    add() {
3      this.containerService.add({name: 'centos'});
4    }
5
6    delete() {
7      this.containerService.delete(1);
8    }
9
10   update() {
11     this.containerService.update({id: 1, name: 'ubuntu:18.03'});
12   }
13
14
15   getById(id: number) {
16     this.containerService.getByKey(id);
17   }
18
19 }
```

CRUD

```
1  export class ContainersComponent implements OnInit {
2    add() {
3      this.containerService.add({name: 'centos'});
4    }
5
6    delete() {
7      this.containerService.delete(1);
8    }
9
10   update() {
11     this.containerService.update({id: 1, name: 'ubuntu:18.03'});
12   }
13
14
15   getById(id: number) {
16     this.containerService.getByKey(id);
17   }
18
19 }
```

CRUD

https://stackblitz.com/edit/ngrx-data-demo-3?embed=1&file=src/app/containers/containers.component.ts&hideExplorer=0

Flow: getAll

https://ngrx-data.firebaseapp.com/index.html?auto=false&program=ngrxDataGe

Action/HttpRequest的默认规则

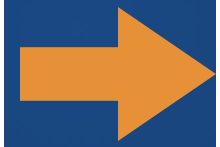
Operation	action.type	http request url
getAll	[Container] @ngrx/data/query-all	GET api/containers
add	[Container] @ngrx/data/save/add-one	POST api/containers
update	[Container] @ngrx/data/save/update-one	PUT api/containers
delete	[Container] @ngrx/data/save/delete-one	DELETE api/containers
getByKey	[Container] @ngrx/data/query-by-key	GET api/containers/${key}
getWithQuery	[Container] @ngrx/data/query-many	GET api/containers?query

@ngrx/entity

```
{
  entityCache: {
    Container: {
      ids: [
        1,
        2,
        3,
        4
      ],
      entities: {
        '1': {
          id: 1,
          name: 'ubuntu'
        },
        '2': {
          id: 2,
          name: 'alpine'
        }
      }
    }
  }
```

@ngrx/entity的好处

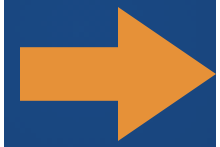
```
Containers: [
  {
    id: 1,
    name: 'ubuntu'
  },
  {
    id: 2,
    name: 'alpine'
  }
]
```



```
Containers: {
  ids: [1, 2],
  entities: {
    1: {
      id: 1,
      name: 'ubuntu'
    },
    2: {
      id: 2,
      name: 'alpine'
    }
  }
}
```

@ngrx/entity的好处

```
Containers: [
  {
    id: 1,
    name: 'ubuntu'
  },
  {
    id: 2,
    name: 'alpine'
  }
]
```



```
Containers: {
  ids: [1, 2],
  entities: {
    1: {
      id: 1,
      name: 'ubuntu'
    },
    2: {
      id: 2,
      name: 'alpine'
    }
  }
}
```

@ngrx/entity提供的EntityAdpter

- 内建CRUD方法支持数据转换成需要的格式
- 提供了基础的Selectors
- 提供了可供定制的idSelector和sortComparer

ngrx/data -> ngrx/entity

https://ngrx-data.firebaseapp.com/index.html?auto=false&program=ngrxDataEntity

Filter

```
1  // define a filter function and register it to entityMetadata
2
3  export function containerNameFilterFn(
4    containers: ContainerUIModel[],
5    pattern: string) {
6    return PropsFilterFnFactory(['name'])(containers, pattern);
7  }
8
9  export const entityMetadata: EntityMetadataMap = {
10   Container: {
11     filterFn: containerNameFilterFn
12   }
13 }
14
15 // Containers Component
16 export class ContainersComponent {
17   this.filteredContainers$ = this.containerService.filteredEntities$;
18 }
```

Filter

```
1  // define a filter function and register it to entityMetadata
2
3  export function containerNameFilterFn(
4    containers: ContainerUIModel[],
5    pattern: string) {
6    return PropsFilterFnFactory(['name'])(containers, pattern);
7  }
8
9  export const entityMetadata: EntityMetadataMap = {
10   Container: {
11     filterFn: containerNameFilterFn
12   }
13 }
14
15 // Containers Component
16 export class ContainersComponent {
17   this.filteredContainers$ = this.containerService.filteredEntities$;
18 }
```

Sort

```
1  // define a compare function and register it to entityMetadata
2
3  export const entityMetadata: EntityMetadataMap = {
4    Container: {
5      sortComparer: (c1: Container, c2: Container) => c1.id - c2.id
6    }
7  }
8
9  // Containers Component
10 export class ContainersComponent {
11   this.containers$ = this.containerService.entities$;
12 }
```

Sort

```
1  // define a compare function and register it to entityMetadata
2
3  export const entityMetadata: EntityMetadataMap = {
4    Container: {
5      sortComparer: (c1: Container, c2: Container) => c1.id - c2.id
6    }
7  }
8
9  // Containers Component
10 export class ContainersComponent {
11   this.containers$ = this.containerService.entities$;
12 }
```

idSelector

```
1 // define a idSelector function and register it to entityMetadata
2
3 export const entityMetadata: EntityMetadataMap = {
4   Container: {
5     selectId: (c: Container) => c1.fingerprint
6   }
7 }
8
```

idSelector

```
1 // define a idSelector function and register it to entityMetadata
2
3 export const entityMetadata: EntityMetadataMap = {
4   Container: {
5     selectId: (c: Container) => c1.fingerprint
6   }
7 }
8
```

Loading

```
1 // Containers Component
2 export class ContainersComponent {
3   this.loading$ = this.containerService.loading$;
4 }
5
6 // Containers Component template
7 <ng-container *ngIf="loading$ | async; else containers_tpl">
8   Loading...
9 </ng-container>
```

Loading

```
1 // Containers Component
2 export class ContainersComponent {
3   this.loading$ = this.containerService.loading$;
4 }
5
6 // Containers Component template
7 <ng-container *ngIf="loading$ | async; else containers_tpl">
8   Loading...
9 </ng-container>
```

Demo: Filter/Sort/Loading

https://stackblitz.com/edit/ngrx-data-demo-4?embed=1&file=src/app/containers/containers.component.ts&hideExplorer=0

乐观更新

- 单独的流程处理先更新客户端持久状态，再更新服务器状态
- 要做错误处理，当有错误的时候能够支持回滚或者重试
- 为了支持回滚和重试，本地还需要存储变更的内容(ChangeSet)
- 回滚时候也要考虑其他操作是否已经更改了数据

使用@ngrx/data实现乐观更新

```
export const entityMetadata: EntityMetadataMap = {
  Container: {
    entityDispatcherOptions: {
      optimisticAdd: false,
      optimisticUpdate: true,
      optimisticDelete: true
    }
  }
};
```

使用@ngrx/data实现乐观更新

```
export const entityMetadata: EntityMetadataMap = {
  Container: {
    entityDispatcherOptions: {
      optimisticAdd: false,
      optimisticUpdate: true,
      optimisticDelete: true
    }
  }
};
```

ChangeState

```
changeState: {
  '2': {
    changeType: 3,
    originalValue: {
      id: 2,
      name: 'alpine'
    }
  }
}
```

错误处理

```
@Injectable({ providedIn: 'root' })
export class ErrorService {
  constructor(private actions$: Actions) {
    actions$
      .pipe(
        ofEntityOp(),
        filter(
          (ea: EntityAction) =>
          ea.payload.entityOp.endsWith(OP_ERROR)
        )
      )
      // this service never dies so no need to unsubscribe
      .subscribe(action => {
        if (action.payload.entityOp.endsWith(OP_ERROR)) {
          this.containerService.createAndDispatch(EntityOp.UNDO_ALL);
        }
      });
  }
}
```

错误处理

```
@Injectable({ providedIn: 'root' })
export class ErrorService {
  constructor(private actions$: Actions) {
    actions$
      .pipe(
        ofEntityOp(),
        filter(
          (ea: EntityAction) =>
          ea.payload.entityOp.endsWith(OP_ERROR)
        )
      )
      // this service never dies so no need to unsubscribe
      .subscribe(action => {
        if (action.payload.entityOp.endsWith(OP_ERROR)) {
          this.containerService.createAndDispatch(EntityOp.UNDO_ALL);
        }
      });
  }
}
```

Demo: Update

https://stackblitz.com/edit/ngrx-data-demo-5?embed=1&file=src/app/containers/containers.component.ts&hideExplorer=0

完美！！！

看来不需要学习太多ngrx的知识也不用写各种ngrx的bolierplate就可以得到所有ngrx的好处

但是……

定制Http Request的规则

```
1  // create a customize http url generator
2  @Injectable({providedIn: 'root'})
3  export class CustomizeHttpUrlGenerator extends DefaultHttpUrlGenerator
4    protected getResourceUrls(
5      entityName: string,
6      root: string
7    ): HttpResourceUrls {
8      const urls = super.getResourceUrls(entityName, root);
9      urls.entityResourceUrl = urls.collectionResourceUrl;
10     return urls;
11   }
12 }
13
14 // app.module.ts, then register to providers
15 {
16   provide: HttpUrlGenerator,
17   useClass: CustomizeHttpUrlGenerator
18 }
```

定制Http Request的规则

```
1  // create a customize http url generator
2  @Injectable({providedIn: 'root'})
3  export class CustomizeHttpUrlGenerator extends DefaultHttpUrlGenerator
4    protected getResourceUrls(
5      entityName: string,
6      root: string
7    ): HttpResourceUrls {
8      const urls = super.getResourceUrls(entityName, root);
9      urls.entityResourceUrl = urls.collectionResourceUrl;
10     return urls;
11   }
12 }
13
14 // app.module.ts, then register to providers
15 {
16   provide: HttpUrlGenerator,
17   useClass: CustomizeHttpUrlGenerator
18 }
```

服务器和客户端的数据结构不一致，需要Mapping

```
// customize
@Injectable({ providedIn: 'root' })
export class CustomizeDataService extends DefaultDataService<Container> {
  constructor(http: HttpClient, httpUrlGenerator: HttpUrlGenerator, logger:
    super('Container', http, httpUrlGenerator);
  }
  getAll(): Observable<Container[]> {
    return super.getAll().pipe(
      map(containers =>
        containers.map(c => ({...c, fullName: c.name + c.tag)}))
      )
    );
  }
}
```

服务器和客户端的数据结构不一致，需要Mapping

```
// customize
@Injectable({ providedIn: 'root' })
export class CustomizeDataService extends DefaultDataService<Container> {
  constructor(http: HttpClient, httpUrlGenerator: HttpUrlGenerator, logger:
    super('Container', http, httpUrlGenerator);
  }
  getAll(): Observable<Container[]> {
    return super.getAll().pipe(
      map(containers =>
        containers.map(c => ({...c, fullName: c.name + c.tag)}))
      )
    );
  }
}
```

分页处理

```
entityCache: {
  Container: {
    ids: [],
    entities: {},
    entityName: 'Container',
    filter: '',
    loaded: true,
    loading: false,
    changeState: {}
  }
}

// entity metadata.ts

export const entityMetadata: EntityMetadataMap = {
  Container: {
    additionalCollectionState: {
      page: {//...      }
    },
  }
};
```

分页处理

```
entityCache: {
  Container: {
    ids: [],
    entities: {},
    entityName: 'Container',
    filter: '',
    loaded: true,
    loading: false,
    changeState: {}
  }
}

// entity metadata.ts

export const entityMetadata: EntityMetadataMap = {
  Container: {
    additionalCollectionState: {
      page: {//...      }
    },
  }
};
```

不知道该如何用
@ngrx/data来实现

最终

- 重头系统学习ngrx
- 学习@ngrx/data的文档和源代码

@ngrx/data扩展点

- 定制@ngrx/data默认提供的行为
  - 定制 EntityCollectionService
  - 定制 EntityAction/Dispatcher
  - 增加属性到 EntityCollection
  - 定制 DataService
  - 定制 存储时候合并的策略
  - 定制 http url 规则
  - 定制 复数名称
  - 定制 存储行为
- 完全和普通的ngrx可以并存
  - 可以脱离@ngrx/data写自己的
  - Action/Reducer/Effects/Selector

分页

https://ngrx-data.firebaseapp.com/index.html?auto=false&program=ngrxDataPag

具体实现代码

```
1  // entity metadata
2  EntityMetadataMap = {
3    Container: {
4      additionalCollectionState: { page: {} }
5    }
6  };
7  // Custom Data Service to save page from backend to data
8  export class CustomizeDataService extends DefaultDataService<Container> {
9    getWithQuery(params: string | QueryParams): Observable<Container[]> {
10     const pageIndex = params['pageIndex'] || '1';
11     return this.http.get(`api/containers?pageIndex=${pageIndex}`).
12       pipe(map((data: any) => {
13         const containers = data.data;
14         containers.page = data.page;
15         return containers as any;
16     }));
17   }
18 }
19 // create a page$ in service
20 export class ContainerService extends EntityCollectionServiceBase<Container> {
21   page$: Observable<{pageIndex: number, pageCount: number}>;
22   constructor(serviceElementsFactory: EntityCollectionServiceElementsFactory,
23     private http: HttpClient) {
24     super('Container', serviceElementsFactory);
25
26     this.page$ = this.selectors$['page$'];
27   }
28
```

具体实现代码

```
1  // persistent handler
2  @Injectable({ providedIn: 'root' })
3  export class PagePersistenceResultHandler extends DefaultPersistenceResultHandler {
4    handleSuccess(originalAction: EntityAction): (data: any) => Action {
5      const actionHandler = super.handleSuccess(originalAction);
6      return (data: any) => {
7        const action = actionHandler(data);
8        if (action && data && data.page) {
9           (action as any).payload.page = data.page;
10       }
11       return action;
12     };
13
14   }
15 }
16 // reducers methods.
17 export class EntityCollectionPageReducerMethods<T> extends EntityCollectionReducerMethods<T>
18   constructor(public entityName: string, public definition: EntityDefinition<T>) {
19     super(entityName, definition);
20   }
21
22   protected queryManySuccess(collection: EntityCollection<T>,action: EntityAction<T[]>
23   ): EntityCollection<T> {
24     const ec = super.queryManySuccess(collection, action);
25     if ((action.payload as any).page) {
26       (ec as any).page = (action.payload as any).page;
27     }
28     return ec;
29   }
30 }
```

Demo: 分页

https://stackblitz.com/edit/ngrx-data-demo-7?embed=1&file=src/app/containers/containers.component.ts&hideExplorer=0

根据条件的错误处理

```
1  // ComponentA, want to show error dialog
2  this.containers$ = this.containerService.getAll().pipe(
3    catchError(err => {showError(err); return of(err);})
4  );
5  // ComponentB, do not want to show error dialog
6  this.containers$ = this.containerService.getAll().pipe(
7    catchError(err => {log.error(err); return of([])})
8  );
9  // common error service?
10 export class ErrorService {
11   constructor(private actions$: Actions) {
12     actions$
13       .pipe(
14         ofEntityOp(),
15         filter((ea: EntityAction) =>
16           ea.payload.entityOp.endsWith(OP_ERROR)
17         )
18       )
19       .subscribe(action => {
20         showError(...);
21       });
22   }
23 }
```

根据条件的错误处理

```
1  // ComponentA, want to show error dialog
2  this.containers$ = this.containerService.getAll().pipe(
3    catchError(err => {showError(err); return of(err);})
4  );
5  // ComponentB, do not want to show error dialog
6  this.containers$ = this.containerService.getAll().pipe(
7    catchError(err => {log.error(err); return of([])})
8  );
9  // common error service?
10 export class ErrorService {
11   constructor(private actions$: Actions) {
12     actions$
13       .pipe(
14         ofEntityOp(),
15         filter((ea: EntityAction) =>
16           ea.payload.entityOp.endsWith(OP_ERROR)
17         )
18       )
19       .subscribe(action => {
20         showError(...);
21       });
22   }
23 }
```

使用标签

```
// ComponentA, want to show error dialog
this.containers$ = this.containerService.getAll({tag: SHOW_ERROR});
// ComponentB, do not want to show error dialog
this.containers$ = this.containerService.getAll({tag: NO_ERROR});
// common error service?
export class ErrorService {
  constructor(private actions$: Actions) {
    actions$
      .pipe(
        ofEntityOp(),
        filter(
          (ea: EntityAction) =>
          ea.payload.entityOp.endsWith(OP_ERROR)
        )
      )
      .subscribe(action => {
        if (action.payload.tag === SHOW_ERROR) {
          showError(...);
        }
      });
  }
}
```

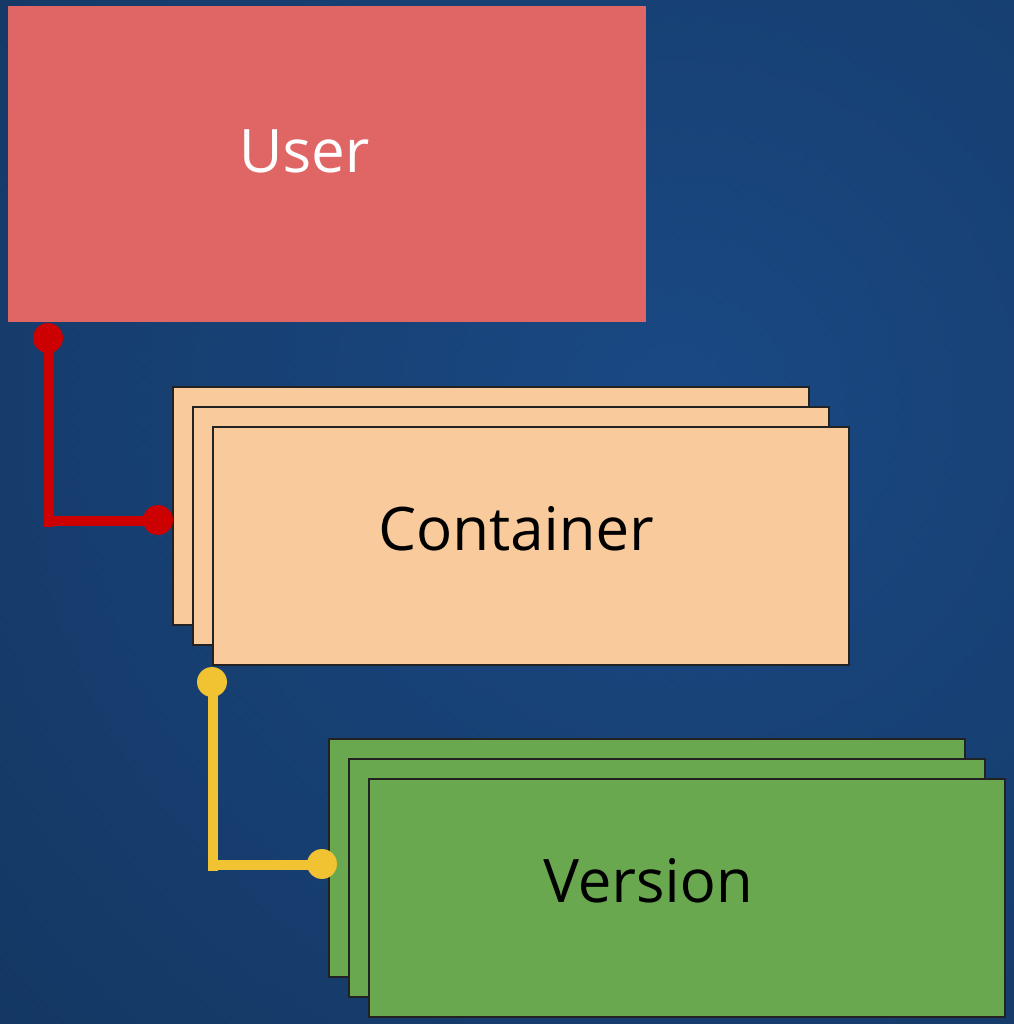
使用标签

```
1  // ComponentA, want to show error dialog
2  this.containers$ = this.containerService.getAll({tag: SHOW_ERROR});
3  // ComponentB, do not want to show error dialog
4  this.containers$ = this.containerService.getAll({tag: NO_ERROR});
5  // common error service?
6  export class ErrorService {
7    constructor(private actions$: Actions) {
8      actions$
9        .pipe(
10         ofEntityOp(),
11         filter(
12           (ea: EntityAction) =>
13           ea.payload.entityOp.endsWith(OP_ERROR)
14         )
15       )
16       .subscribe(action => {
17         if (action.payload.tag === SHOW_ERROR) {
18           showError(...);
19         }
20       });
21   }
22 }
```

Demo: Tag

https://stackblitz.com/edit/ngrx-data-demo-8?embed=1&file=src/app/containers/containers.component.ts&hideExplorer=0

处理有关联关系的实体



@ngrx/data 目前还不支持自动处理有关联关系的实体

```
1  // service
2  export class ContainerService {
3    // action to get containers with versions
4    getContainerWithVersions(containerId) {
5      this.getByKey(containerId).switchMap(container => {
6        this.versionService.getWithQuery({containers: containers.map(con
7          .pipe(map(versions => ({...container, versions})))
8      });
9    }
10
11   // selectors
12   getContainerWithVersionsSelector(containerId) {
13     return this.store.pipe(
14       this.selectors.selectEntities,
15       this.versionService.selectors.selectEntities,
16       (containers, versions) => {
17          const con = containers.find(c => c.id === containerId);
18          const versions = versions.filter(v => vi.cid === containerId)
19          return con ? {...con, versions} : undefined;
20       }
21     );
```

校验?

```
export const entityMetadata: EntityMetadataMap = {
  Container: {
    validations: {
      props: {
        name: ValidatorOptions.required
      },
      validator: (model: T) => ValidationResult
    }
  }
};
```

服务器和客户端数据结构的转换

```
1  // server data structure
2  interface Container {
3    name: string;
4    tag: string;
5    updatedTimeStamp?: string;
6  }
7
8  // frontend data structure
9  interface Container {
10   name: string;
11   tag: string;
12   fullName: string;
13 }
14
15 // entity metadata
16 export const entityMetadata: EntityMetadataMap = {
17   Container: {
18     mapTo: (backendModel) => {}
19   }
20 };
21
```

在我深入学习并解决分页定制问题之前@ngrx/data是一个

- ***一个有效减少ngrx boilerplate代码的库***
- 提供了很多功能处理了应用中出现的常用场景

经历了学习之后!

- *一个使用ngrx并运用了ngrx的最佳实践的库*
  - *命名规则*
  - *模块划分*
  - *扩展设计*
  - *对于RxJs的运用*
- *我们仍然需要掌握ngrx才能彻底掌握@ngrx/data*

如果开发团队是ngrx的经验者

- 可以读一遍@ngrx/data的代码，可以从中学到很多设计的思路!!!
- 可以在某些管理画面使用@ngrx/data

如果开发团队是ngrx的初学者

- 可以把@ngrx/data作为入门库，学习使用ngrx,然后进行简单定制，最后可以完整使用ngrx

我现在的想法

- **@ngrx/data不光可以做简单应用，如果对于ngrx有比较深入了解后，@ngrx/data非常适合作为一个独立的解决方案**

谢谢大家！